01TXFSM - Machine Learning and Deep Learning

Final Project
First Person Action Recognition

Eros Fanì - s269781
Politecnico di Torino
eros.fani@studenti.polito.it

Gabriele Trivigno - s276807
Politecnico di Torino
gabriele.trivigno@studenti.polito.it

Cristiano Gerbino - s277058
Politecnico di Torino
s277058@studenti.polito.it

Abstract

1. Introduction

1.1. Goals

The first goal of the project is to replicate some of the experiments performed in [3] and [2]. The objective of these studies is the First Person Action Recognition: they tried to implement a deep learning model capable to extract meaningful features to automatically predict the action filmed by a wearable camera.

After having replicated these experiments we performed a grid search on the experiments to find the best set of values for the hyperparameters.

At last we have tried to improve the performances of the results of [3] and [2] with some innovative ideas.

1.2. Our contribution

...

1.3. Data exploration

The dataset under analysis is a modified version of GTEA61[1]. The dataset contains the videos in form of frames, and also two kind of preprocessed images: *motion maps* and *optical flows*. The folder schema of the dataset is shown in Figure 1. Videos represent 61 class actions performed by 4 different users (*S1*, *S2*, *S3*, *S4*). Sometimes for some actions more than one video is available. The total number of videos in the dataset is, however, 457, which actually means that it is a quite small dataset.

The optical flow methods try to calculate the motion between two image frames which are taken at times t and $t + \Delta t$ at every voxel position. The warp flow methods try also to remove the motion of the wearable camera. We have two kind of these last representations in our dataset: one computed in the horizontal axis (folder *flow_x_processed*) and one other computed in the vertical axis (folder *flow_y_processed*).

The motion maps are special black-and-white images which represent the spatial location in which the Motion Segmentation task of [2] focuses its attention per each frame. The mmaps present large similarities with the warp flows.

[1] Georgia Tech Egocentric Activity Datasets: http://cbs.ic.gatech.edu/fpv/

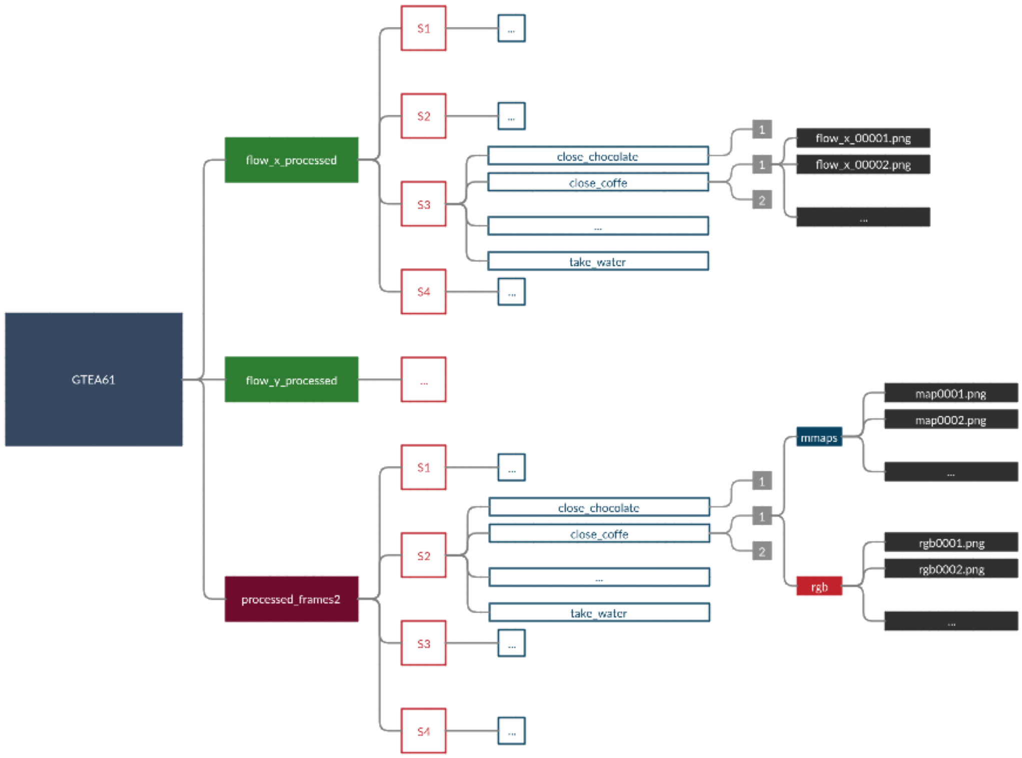

Figure 1: Folder schema of our GTEA61

The differences between the kind of available images in our dataset are shown in Figure 2.

1.4. Data cleaning

The dataset was almost clean already from the beginning, but we encountered two problems within it:

- there were hidden useless folders *.DSstore* inside each one of the user folders. These have been removed
- some of the first mmaps of some videos were missing. In these cases we have simply duplicated the second mmap

2. Descriptions of the models

Here we describe the models that we have used to perform our experiments.

2.1. Egornn

Egornn is a Recurrent Neural Network. The overall architecture of *Egornn* is shown in Figure 3. This net is based on *resnet34*[1], which constitutes the main block. *resnet34* has five convolutional layers inside itself: with respect to Figure 3 they are: *Conv*, *Layer1*, *Layer2*, *Layer3* and , *Layer4*. From now on we will refer to these blocks respectively *conv1*, *conv2*, *conv3*, *conv4* and *conv5*.

At the termination of the *resnet34* is placed a *Spatial Attention Layer*. It includes a *Class Activation Map* (CAM) that is capable to identify the image regions that have been used by the CNN to identify

Figure 2: Types of images in our dataset. In this example is shown a sample of images from the *close_chocolate* action. From the left column to the right column: rgbs, warp flows x, warp flows y, motion maps

the class under analysis. It is computed by taking the output of the *softmax* layer and the output of *conv5* and taking the linear combination of all the weights of *conv5* and the weights of the softmax.

The output of the CAM is then sent to a *softmax* layer to obtain a probability map, which is called *Spatial Attention Layer* (SAM). The output of the SAM is finally multiplied, cell by cell (Hadamard product), with the output of *conv5*, obtaining another tensor of weights which is sent to a *Convolutional Long Term Support Memory* block (ConvLSTM).

The reason for the usage of the ConvLSTM block is that, up to now, what the net does is to take each frame and to try to make predictions based only on the features that the net can extract from those frames, without taking into consideration the temporal encoding of frame level features. The convLSTM block take into consideration, for each frame i, both the output of the SAM for the layer i and the output of the ConvLSTM for the layer $i-1$, constituting a recursive structure.

The last output of the ConvLSTM (the output obtained from the last frame of a particular video) is average pooled and reshaped to obtain a final classification layer with 61 neurons (i.e. the number of classes of our dataset).

2.2. Flow_resnet34

Flow_resnet34 is just a *resnet34* edited to work with the warp flows. It gets five warp flows from *processed_frames_x* and five from *processed_frames_y* in form of a tensor of ten channels and tries to make predictions on the 61 classes.

2.3. Two stream model

Egornn learns appearence features, while *flow_resnet34* learns motion features. The way to join the two nets is to concatenate the two output layers and to add at the end a fully connected layer to get the class category scores.

2.4. Motion Segmentation branch applied to *egornn*

The problem which [2] tries to overcome is that in the two stream model motion and appearence are actually separately learned, without taking into account the spatial-temporal relationships.

We have built an architecture similar to *sparnet*, where the *motion segmentation block* is the same but the *action recognition block* has been substituted by *egornn* (like in one of the attempts in [2]). The architecture is shown in Figure 4. We have

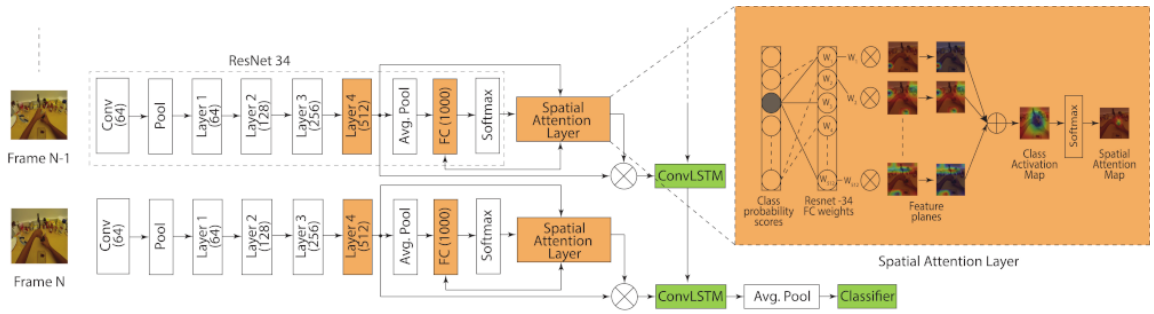

Figure 3: Architecture of *egornn*

used this architecture with some granular variations during our experiments, but the main blocks are always as shown in Figure 4. The input of the convolutional layer of MS Block is taken from one of the convolutional layers of *resnet34* of egornn (the actual layer varies with our experiments). Then, after the convolutional layer, there is a fully connected layer followed by a softmax which normalizes the weights between 0 and 1. *mmaps ground truth* and *rgb ground truth* represent the mmaps and the rgb after the trasformations. The trasformations applied to the mmaps are the same applied to the rgbs, plus a small amount of proper mmaps trasformations which always ends with a trasformation which linearizes the pixels (from a 2 dimensional tensor per mmap we get a 1 dimensional tensor per mmap). For the msblock these linearized pixels represent a real ground truth, because each of the output neurons of the MS Block is used to predict the values of the mmaps ground truth. The pixel losses are summed together (obtaining as result L_{ms}) and then are summed again with the *egornn* loss (L_c). The final loss is used to compute the gradients to update the weights.

2.5. Static-dynamic discriminator

Starting from the model described above, we have added a final binary classifier to *egornn* after the convLSTM, parallel to the other classifier already present which still tries to predict the actual class of the video. The idea was to force the net to learn the motion features from the rgbs. During the training phase this classifier gets 2 kind of sequences of frames: one is the same of the original classifier, while the other gets a sequence of identical frames. This discriminator should be able to recognize the actual videos from the static frames. In this way the gradients should adapt to focus the attention on the motion.

3. Experiments

Our nets are always trained on a predefined train set, which includes all and only the videos of the users *S1*, *S3* and *S4*, while validation and test sets coincide and is constituted by all and only the videos of a single user, *S2*. In addition, the weights of the *resnet34* are pretrained on ImageNet. Each model is always validated while it is trained, so for each training phase we selected the weights with the highest accuracy at a particular epoch as the best ones.

Due to Colab limitations of GPU memory, we have only been able to perform experiments on a limited amount of frames (7 or, in less cases, 16). Due to this problem our results should be interpreted not as absolute value of the accuracy, but as a sort of relative value with respect to the number of frames for each video in our batches.

The size of our batches has always been left to 32, as well as the number of hidden units of the convLSTM module, fixed at 512. Our optimization algorithm is always Adaptive moment estimation (ADAM) with the only exception of *flow_resnet34*, for which it is Stochastic Gradient Descent (SGD). When using this last optimizer, the momentum has always been left to 0.9. The scheduler is a MultiStepLR scheduler, which decreases the original learing rate LR by a factor GAMMA at each value of STEP_SIZE.

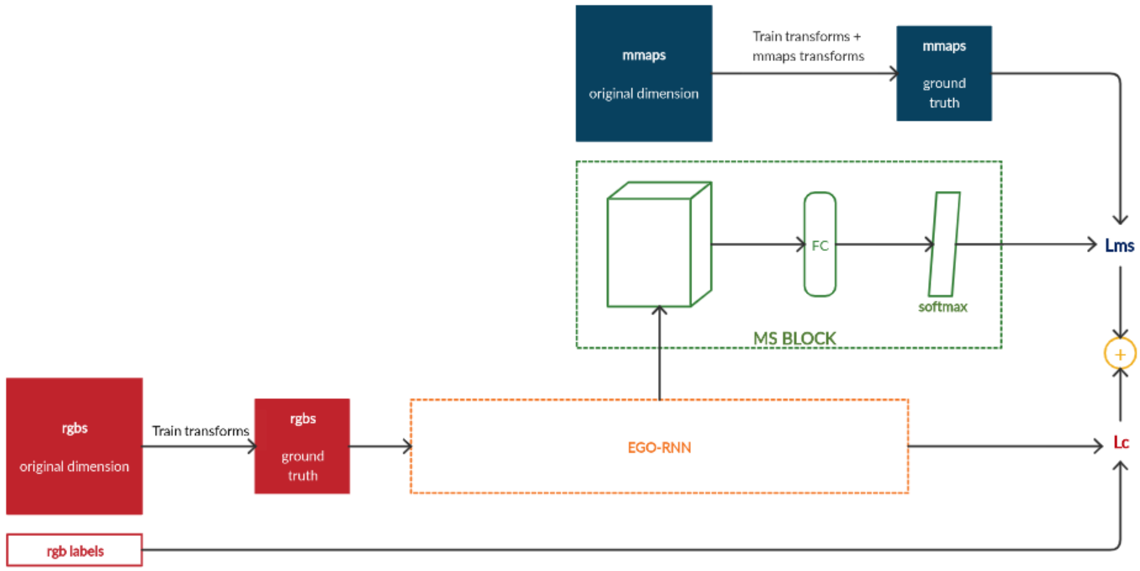

Figure 4: Generic architecture of motion segmentation branch applied to *egornn*

3.1. Egornn

We have replied some of the same experiments of [3] on the original egornn. We have run each of these experiments three times and then we have averaged the results.

First, we have performed the classification by using the *egornn* without and with the CAM. The training phase has been divided in two parts, as in the original paper:

1. train of ConvLSTM and Classifier (green blocks in Figure 3)
2. train of conv5 (layer4 of *resnet34*), FC(1000), Spatial Attention Layer (orange blocks in Figure 3) in addition to the previously listed blocks

The values of the hyperparameters for the first stage are:

LR	1e-3
WEIGHT_DECAY	4e-5
NUM_EPOCHS	200
STEP_SIZE	[25, 75, 150]
GAMMA	0.1

While, for the second stage, they are:

LR	1e-4
WEIGHT_DECAY	4e-5
NUM_EPOCHS	150
STEP_SIZE	[25, 75]
GAMMA	0.1

Then, we have also trained *flow_resnet34* alone. In this case we used only 5 frames per each flow (x and y) due to the fact that for some videos no more than 5 frames were provided.

The values of the hyperparameters in this case are:

LR	1e-2
WEIGHT_DECAY	5e-4
NUM_EPOCHS	750
STEP_SIZE	[150, 300, 500]
GAMMA	0.5

At last we performed the two stream training with the following values for the hyperparameters:

Configurations	Frames	Mean accuracy
EGO-RNN without CAM - stage 1	7	29.89
EGO-RNN without CAM - stage 1	16	27.87
EGO-RNN without CAM - stage 2	7	50.00
EGO-RNN without CAM - stage 2	16	50.57
EGO-RNN - stage 1	7	41.38
EGO-RNN - stage 1	16	46.84
EGO-RNN - stage 2	7	58.91
EGO-RNN - stage 2	16	65.52
flow_resnet34	5	46.26
two-stream (joint train)	7*	57.76
two-stream (joint train)	16*	66.38

Figure 5: Summary of the results over different configurations. Each value of the mean accuracy is the mean of the accuracies over three identical experiments. *the number of frames refers to the *egornn* branch (for the flow_resnet34 branch the number of frames is always 5)

LR	0.99
LR_FLOW	1e-4
WEIGHT_DECAY	5e-4
NUM_EPOCHS	250
STEP_SIZE	[1]
GAMMA	0.5

Where LR is the learning rate of *egornn* and LR_FLOW is the learning rate of *flow_resnet34*.

The summary of our results is shown in Figure 5. From here it raises that the best model is the two-stream (joint train) with 16 frames, followed by EGO-RNN - stage 2 with 16 frames, which is behind the two-stream model by less than 1 point of mean accuracy. Due to the averaging between three identical runs we can rely on this result and assert that the contribution of *flow_resnet34* slightly increases the performances, but also that the most of the contribution is given by *egornn*.

In Figure 6 and Figure 7 are shown respectively the validation accuracy and the validation loss by epoch of one random extracted run per each one of the attempts with 16 frames (5 in case of *flow_resnet34*) and only for the stage 2 when a two stage training is requiered.

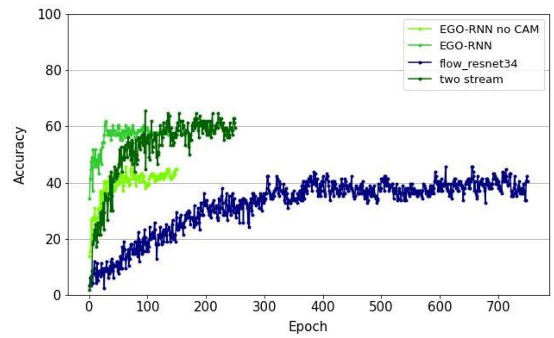

Figure 6: Validation accuracy by epoch of one random extracted run for the four most interesting training configurations

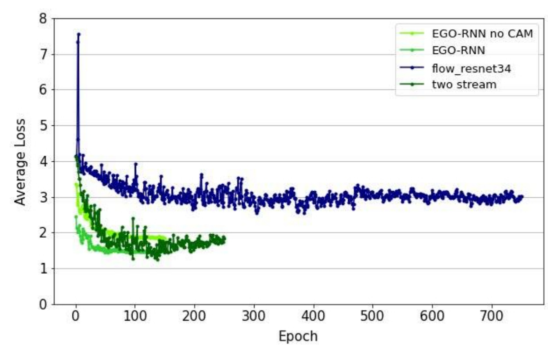

Figure 7: Validation loss by epoch of one random extracted run for the four most interesting training configurations. The losses are the average of each batch loss within a single epoch

From Figure 6 and Figure 7 is even more evident that *flow_resnet34* is highly inefficient alone, and that the results with the CAM are heavily better than the results without the CAM (higher accuracy and lower loss at every epoch). The two-stream model requieres more time to get high accuracies, and overall it seems to have the same behaviour of *egornn* when at full capacity, but it is noisier and so it is easier that for some epoch the accuracy is higher.

Stage	Frames	Mean accuracy
1	7	44.83
1	16	50.86
2	7	60.63
2	16	62.07

Figure 8: Mean accuracies over three identical experiments with the same fixed set of values for the hyperparameters, by varying only the stage of training and the number of frames - classification experiment

3.2. Motion Segmentation branch applied to egornn

First of all we have replied the same experiment of [2] (the one in which *egornn* is the Action Recognition Block). The hyperparameters used are:

LR	1e-3
WEIGHT_DECAY	4e-5
NUM_EPOCHS	150
STEP_SIZE	[50, 100]
GAMMA	0.1

for the stage1, while they are the following:

LR	1e-4
WEIGHT_DECAY	4e-5
NUM_EPOCHS	150
STEP_SIZE	[25, 75]
GAMMA	0.1

for the stage2. We have decreased the number of epochs for the stage1 because we have observed that going too far with the epochs, with the loss decreased after the steps of the optimizer, would be meaningless and does not give any significant result.

The results with the same values for the hyperparameters are shown in Figure 8.

As expected, the best performances are achieved after the second stage of training and with 16 frames.

We have also replied this experiment as a regression problem. [REGRESSION IMPLEMENTATION]

Stage	Frames	Mean accuracy
1	7	44.83
	16	50.86
2	7	59.77
	16	66.38

Figure 9: Mean accuracies over three identical experiments with the same fixed set of values for the hyperparameters, by varying only the stage of training and the number of frames - regression experiment

In Figure 9 are shown the results.

Also in this case the highest value of the accuracy is obtained for 16 frames. The first stage is exactly the same so we have not replied it three times more. For the second stage we observe that with this combination of values for the hyperparameters the regression performs more than 4 points better than classification with 16 frames, while the accuracies are roughly the same when the number of frames is 7.

We have performed also a complete grid search to improve the performances of the net. The number of selected frames is 16, for which we have seen that the performances are better than for 7. For the classification strategy the results are shown in Figure 10, while for the regression they are shown in Figure 11.

As we can observe the best performances are achieved:

- with $LR = 10^{-4}, WEIGHT\_DECAY = 4 \cdot 10^{-3}, STEP\_SIZE = [30, 80]$ for the classification method, for an accuracy of 75.00
- with $LR = 5 \cdot 10^{-4}, WEIGHT\_DECAY = 4 \cdot 10^{-5}, STEP\_SIZE = [30, 80]$ for the classification method, for an accuracy of 72.41

In general we can say that the performances are better with the classification method, and this is a surprisingly relult based on the observations without the grid. The reason for the previous result was that, as emerges from Figure 10 and Figure 11, the

		GAMMA	0.1	0.2	0.5
LR	WEIGHT_DECAY	STEP_SIZE			
1e-04	4e-03	[30, 70]	73.28	64.66	65.52
1e-04	4e-03	[40, 80]	67.24	75.00	67.24
1e-04	4e-03	[50, 100]	67.24	67.24	68.10
1e-04	4e-05	[30, 70]	62.07	68.97	65.52
1e-04	4e-05	[40, 80]	67.24	68.97	66.38
1e-04	4e-05	[50, 100]	65.52	67.24	67.24
5e-04	4e-03	[30, 70]	68.10	64.66	70.69
5e-04	4e-03	[40, 80]	65.52	62.93	63.79
5e-04	4e-03	[50, 100]	62.45	65.52	66.38
5e-04	4e-05	[30, 70]	64.66	65.52	61.23
5e-04	4e-05	[40, 80]	65.52	66.38	62.07
5e-04	4e-05	[50, 100]	68.97	62.45	67.24

Figure 10: Accuracies at various combinations of hyperparameters for the classification strategy

		GAMMA	0.1	0.2	0.5
LR	WEIGHT_DECAY	STEP_SIZE			
1e-04	4e-03	[30, 80]	62.07	62.93	62.07
1e-04	4e-03	[40, 90]	66.38	68.97	64.66
1e-04	4e-05	[30, 80]	60.34	62.07	67.24
1e-04	4e-05	[40, 90]	62.93	66.38	64.66
5e-04	4e-03	[30, 80]	64.66	61.23	67.24
5e-04	4e-03	[40, 90]	65.52	66.38	71.55
5e-04	4e-05	[30, 80]	68.10	56.03	72.41
5e-04	4e-05	[40, 90]	66.38	65.52	68.10
5e-05	4e-03	[30, 80]	61.21	63.79	62.07
5e-05	4e-05	[30, 80]	61.21	62.07	66.38

Figure 11: Accuracies at various combinations of hyperparameters for the regression strategy

optimal values for the hyperparameters falls in a total different region with respect to the two different methodologies.

In Figure **??** we can observe the effects of the values of the hyperparameters on the accuracy and on the loss.

To overcome these problems we have tried to increase the resolution of the downsampled mmaps and to treat the problem as.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition, 2015.

[2] M. Planamente, A. Bottino, and B. Caputo. Joint encoding of appearance and motion features with self-supervision for first person action recognition, 2020.

[3] S. Sudhakaran and O. Lanz. Attention is all we need: Nailing down object-centric attention for egocentric activity recognition, 2018.

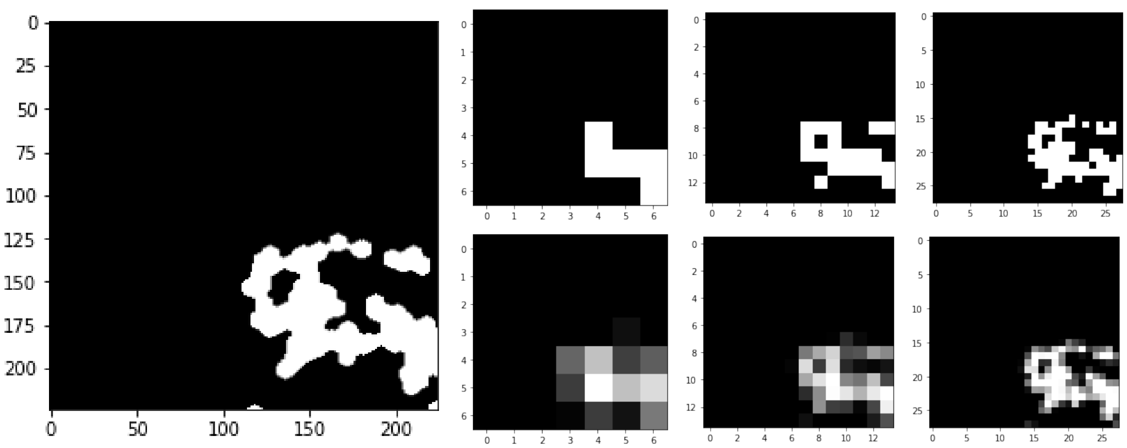

Figure 12